

# IEEE Copyright Notice

Copyright (c) 2014 IEEE

Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Published in Proceedings of 5<sup>th</sup> International Conference on Advanced Computing & Communication Technologies (ACCT-2015), Feb. 21-22, 2015, Rohtak, India

DOI: 10.1109/ACCT.2015.114

# Developing Postfix-GP Framework for Symbolic Regression Problems

Vipul K. Dabhi

Department of Information Technology,  
Faculty of Technology, Dharmsinh Desai University, Nadiad, Gujarat, INDIA  
`vipul.k.dabhi@gmail.com`

Sanjay Chaudhary

Institute of Engineering and Technology,  
Ahmedabad University, Ahmedabad-380009, Gujarat, INDIA  
`sanjay.chaudhary@ahduni.edu.in`

## Abstract

This paper describes Postfix-GP system, postfix notation based Genetic Programming (GP), for solving symbolic regression problems. It presents an object-oriented architecture of Postfix-GP framework. It assists the user in understanding of the implementation details of various components of Postfix-GP. Postfix-GP provides graphical user interface which allows user to configure the experiment, to visualize evolved solutions, to analyze GP run, and to perform out-of-sample predictions. The use of Postfix-GP is demonstrated by solving the benchmark symbolic regression problem. Finally, features of Postfix-GP framework are compared with that of other GP systems.

*Keywords-* Postfix Genetic Programming; Postfix-GP Framework; Object Oriented Design; GP Software Tool; Symbolic Regression

## 1 Introduction

Evolutionary Algorithms (EA) are a group of computational techniques, which employ the theory of natural selection to a population of individuals to generate better individuals. Genetic Programming (GP) is a paradigm of EA which uses hierarchical, tree structure, variable length representation to code solutions of a problem. GP can be used to intelligently search the solution space for finding the optimal solution of a problem.

There are many GP tools (lil-gp, GeneXproTools, GPLab, ECJ, Open BEAGLE) [1, 2, 3, 4, 5] developed by GP practitioners. However, none of these address the following demands of end user before applying them to solve

symbolic regression problems: (i) ease of use and (ii) small learning curve. Many of these tools are open source and available freely (lil-gp, ECJ, Open BEAGLE, GPLab for MATLAB) [1, 4, 5, 3] whereas the rest are available commercially (GeneXproTools) [2]. Many of these tools necessitate modification in source code in order to generate required experimental environment. Determining the final solution, produced by these tools, demands translation of the output or digging the log files. Due to these reasons, the interest of researchers and engineers in GP may get reduced. These reasons motivated us to develop our own GP framework [6] which uses the postfix notation for an individual representation.

We have considered the following features which need to be supported by Postfix-GP framework: (i) easy to extend, (ii) simple and quick procedure for the configuration and running of GP, (iii) a set of algorithm implementation for: (a) generating the initial population, (b) selection mechanisms, and (c) genetic operators, (iv) visualization of: (a) Postfix-GP run analysis and (b) evolved solution with statistical measures, (v) one-step and multi-step prediction support, (vi) visualization of results for one-step and multi-step predictions, and (vii) storage and retrieval of evolved solutions to and from file. Postfix-GP has been used in experimental work [6], [7], [8], [9].

This paper presents the design and implementation of Postfix-GP, an object oriented software framework for genetic programming. Section 2 gives introduction to GP. Section 3 presents the design of Postfix-GP. Moreover, the section also gives the implementation details and main features of Postfix-GP. Section 4 presents Postfix-GP as a solution modeling tool by solving the benchmark symbolic regression problem. Section 5 compares the features of Postfix-GP with lil-gp [1], ECJ [4], and JCLEC [10] frameworks. This is followed by conclusions in Section 6.

## 2 Introduction to Genetic Programming

Standard GP [11] employs a variable length, tree structure scheme for an individual representation. The tree can be used to represent logical expressions (IF-THEN-ELSE), boolean expressions (AND,OR,NOT) or algebraic expressions. The symbolic regression aims to find the functional relationship (mathematical expression) between given instances of inputs-outputs. GP can be used to perform Symbolic Regression (SR). When using GP for solving symbolic regression problems, the user need to specify the following items: (i) GP configuration parameters, (ii) terminal set and function set, (iii) fitness function.

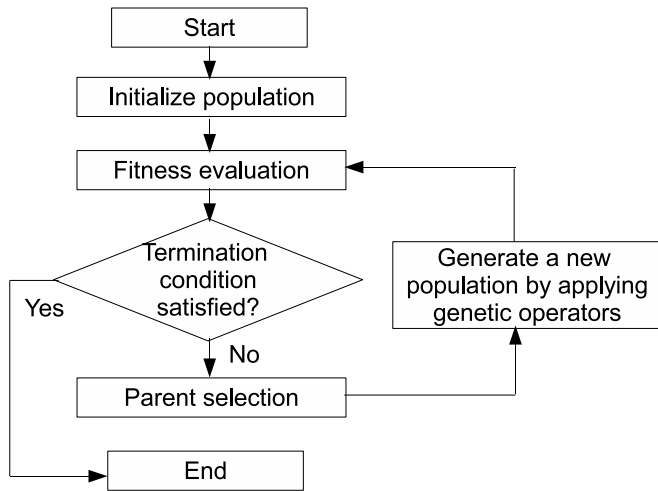


Figure 1: Flowchart of Genetic Programming.

The main steps of standard GP [11] are as follows:

1. random generation of an initial population of candidate solutions in tree form using the elements of function set and terminal set, selected by the modeler (user).
2. calculating fitness value of every individual of the population on the given training dataset (fitness cases)
3. selecting parents for mating based on the calculated fitness values, determined in previous step
4. applying sub-tree crossover and mutation (genetic operators) on selected parents for generating a new population of individuals.

The process is repeated until the termination condition is fulfilled.

## 3 Postfix-GP

### 3.1 Features included in the Design of Postfix-GP

The important features of the proposed Postfix-GP are categorized into: (i) training dataset, function set, and terminal set related, (ii) GP parameters related, (iii) test dataset prediction related, (iv) GP run analysis related, and (v) serialization and de-serialization of GP experiments and results.

Training dataset, function set, and terminal set related features:

- Loading of training dataset
- Loading of binary and unary functions
- Loading of constants

GP parameters related:

- Selection of method to generate initial population
- Configuration of GP parameters like population size, number of generations, crossover rate, mutation rate
- Selection of crossover and mutation type
- Selection of selection scheme

Test dataset prediction related:

- Loading out-of-sample test dataset for one-step predictions
- Visualization of results of one-step predictions with statistical measures
- Loading out-of-sample test dataset for multi-step predictions
- Visualization of results of multi-step predictions with statistical measures

GP run analysis related:

- Plotting best adjusted fitness vs number of generations
- Plotting average adjusted fitness vs number of generations
- Plotting solution size vs number of generations

Serialization and de-serialization of GP experiments and results:

- Serialization of GP parameters, function set, terminal set, and obtained solutions to a file
- De-serialization of GP parameters, function set, terminal set, and obtained solutions from a file

### 3.2 Implementation of Postfix-GP

This section presents the implementation details of Postfix-GP. This will be useful to the reader in understanding and customizing the proposed Postfix-GP framework. For details related to Postfix-GP solution representation scheme, refer [6], [7]. Postfix-GP framework is developed using Microsoft .NET framework [12] on Windows XP operating system. The ZedGraph [13] class library is used for plotting the charts. ZedGraph is an open source graph library for .NET platform.

The class diagram for Postfix-GP is depicted in Figure 2. The classes can be grouped into following categories: (i) representation (Genome, EquationGenome), (ii) population (Population), (iii) crossover operator (BaseCrossover, GA-like, Sub-tree, Semantic aware sub-tree), (iv) mutation operator (BaseMutation, Fully Protected, Partially Protected), (v) selection schemes (BaseSelection, Roulette-wheel, Tournament, Parsimony Pressure), (vi) GP parameters (GP Parameters), and (vii) statistical analysis of results (GP Run Results).

### 3.3 Individual, Population and Fitness

In Postfix-GP, an individual represents a candidate solution for a problem. We can create an instance of individual using EquationGenome class, which is derived from an abstract class called Genome. To generate an individual, we can use two-argument constructor, which returns a new individual having ValidLength in between MinLength and MaxLength.

```
public class EquationGenome() {
    public static int MinLength;
    public static int MaxLength;
    private int ValidLength;
    private ArrayList TheArray = new ArrayList();
    private int Rawfitness;
    private int Adjustedfitness;
    public EquationGenome(int Minlength, int Maxlength)
}
```

An individual comprises a genotype and a phenotype. The genotype is of type array of size equals to MaxLength specified by the user. In addition to the genetic material, an individual also contains raw fitness and adjusted fitness. The raw fitness is not range bound whereas the adjusted fitness can take values in the range of 0 to 1.

In Postfix-GP, an instance of a population can be created using the class Population. An instance of Population contains a collection of instances of EquationGenome. Individuals can be added to and removed from the population (collection) using the iterator. Iterator is useful to iterate over a collection in sequential order. To create an instance of a population, several Postfix-GP parameters need to be passed. Following constructor can be used for the same.

```
public Population(CurrentDirectory, NoofGeneration,
    NoofGenerationperCascade, PopulationSize, MinLength,
    MaxLength, MutationFrequency, CrossoverType, Cross-
    overFrequency, SelectionStrategy, Intervalarithmic,
    SemanticSensitivity, InitialPopulationType, MutationType)
```

The number of elites of a population can be set using archivepopulationsize parameter. By default, the archive size is set to 1/10 of population size. The information related to an instance of a Population can be printed to a log file in human interpretable form. The logged population information is presented below. It begins by printing the number of generation, then details of individuals of current population and archive. For each individual it prints the genotype representation, its ValidLength and adjusted fitness value.

Generation 0

Population

```
a#-0.6#a#a#*#*#-#a#-0.3#0.8#+#-#*# → AdjFit → 0.0011 → ValPos → 13
a#a#-0.8#-#*#a#/#-0.2#a#0.3#-#*#+# → AdjFit → 0.0010 → ValPos → 13
a#-0.6#a#a#*#-0.8#a#a#*#*#*#+#-#-0.7#+# → AdjFit → 0.0002 → ValPos → 15
a#a#-0.4#a#*#-#a#0.1#S#+#*# → AdjFit → 0.0003 → ValPos → 12
```

Archive

```
a#a#-0.4#a#*#-#a#0.1#S#+#*# → AdjFit → 0.0003 → ValPos → 12
a#-0.6#a#a#*#-0.8#a#a#*#*#*#+#-#-0.7#+# → AdjFit → 0.0002 → ValPos → 15
```

Finally, a fitness value is assigned to an individual by evaluating it on training dataset. A code snippet for calculating a fitness of an individual is presented here. The code checks for type of terminal before performing any operation.

```
public float PostFixCalculation(float[] inputs) {
    equationstack.Clear();
```

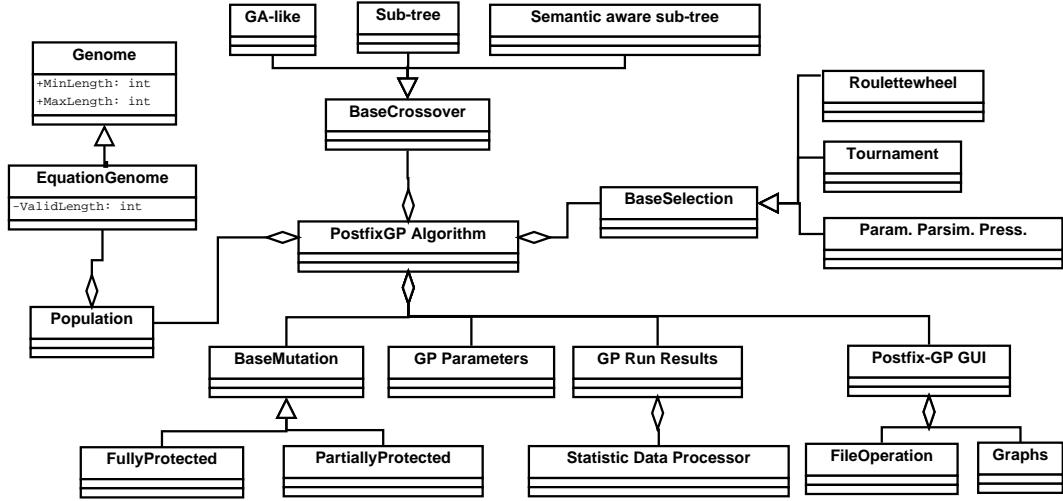


Figure 2: Class diagram for Postfix-GP.

```

for (int i = 0; i < this.ValidLength; i++) {
    if ((int)TheArray[i] < Operandcount) {
        // Push a single input value on stack
        equationstack.Push(inputs[(int)TheArray[i]]);
    }
    else if ((int)TheArray[i] < BinaryoperatorRange) {
        if (equationstack.Count > 0) {
            y = (float)equationstack.Pop();
            if (equationstack.Count > 0) {
                x = (float)equationstack.Pop();
                result=DoOperation(x,y,Binaryoperators[(int)TheArray[i]-
                    OperandRange].ToString());
                if(double.IsPositiveInfinity(result)
                    ||double.IsNegativeInfinity(result)) {
                    return result;
                }
                else
                    equationstack.Push(result);
            }
        }
    }
    else if ((int)TheArray[i] < UnaryoperatorRange) {
        .....
    }
}

```

### 3.4 Selection Methods

The task of selection mechanism is to select individuals from current population and previous (old) archive.

Postfix-GP permits use of different selection mechanism for population and archive. All implemented classes for selection mechanisms are derived from abstract class **BaseSelection**. The derived classes, **Roulettewheel**, **Tournament**, and **Parsimonypressure**, are required to implement the following two methods: **Select()** and **setIndividuals(ArrayList individuals)**. Each time the **Select()** method is called, it returns the index value. The individual at the given index value in the current population or archive is selected as one of the parents for crossover. The implementation of **Select()** method differs from one selection mechanism to another.

**setIndividuals()** method is used to fix whether the individuals are selected from the current population or from the old archive.

```

public int Select();
public void setIndividuals(ArrayList individuals);

```

## 3.5 Genetic Operators

### 3.5.1 Crossover

Postfix-GP provides implementation for following three types of crossover operators: (i) GA-like one-point crossover [14], (ii) sub-tree crossover [14], and (iii) semantic aware sub-tree crossover [7], [14]. The selection of crossover type is done by the user through GUI. The base class for all crossovers is **BaseCrossover**. The derived classes, **GAlike**, **Subtree**, and **Semantic aware subtree**, are required to implement the **Crossover()** method. The maximum number of trials parameter, used by semantic aware crossover, is

by default set to value 20.

```
public abstract class BaseCrossOver {
    private int MaxCrossoverCount = 20;
    abstract public Genome[] Crossover(Genome gene1, Genome
gene2);
}
```

### 3.5.2 Mutation

Postfix-GP provides implementation for following two types of mutation operators: (i) fully protected and (ii) partially protected. The fully protected mutation selects an index value within the range  $[1, ValidLength]$  and changes the element positioned at the selected index value to another element of the same arity. The partially protected mutation selects an index value within the range  $[1, MaxLength]$ . If the selected index value is less than  $MinLength$  then it applies fully protected mutation (an element positioned at selected index is replaced with another element of the same arity only) else it does not apply this constraint (an element positioned at selected index can be replaced with any other element having either same or different arity). The selection of mutation type is done by the user through GUI. The base class for all types of mutation is `BaseMutation`. The derived classes, `FullyProtected` and `PartiallyProtected`, are required to implement `Mutate()` method. The value of `MaxMutationCount` variable decides how many times Postfix-GP should try to ensure that the mutation operator generates a different gene than the original value. The default value for `MaxMutationCount` is set to value 10. The default value for mutation probability of operators (functions set) is set to value 0.6. The code snippet for abstract class `BaseMutation` and `mutate()` method, implemented by `FullyProtected` class, are presented below.

```
public abstract class BaseMutation {
    public int MaxMutationCount = 10;
    private double operatormutatefrequency = 0.6;
    abstract public EquationGenome Mutate (Equation-
Genome gene);
}
```

### 3.6 Sub-trees as Solutions

Postfix-GP derives all sub-trees of an individual having the `Validlength` greater than the `MinLength` and treats the extracted sub-trees as separate solutions (individuals) during the evolutionary process. This approach is useful to GP for exploring large solution search space and

improving GP performance. Moreover, Postfix-GP uses elitism operator to preserve good solutions from one generation to the next generation without being affected by the destructive nature of crossover and mutation. Elitism also provides good building blocks for producing better solutions in successive generations. Postfix-GP preserves a set of highly fit solutions separately from the main population. Postfix-GP implements elitism using a fixed size archive. An archive is used to preserve the good solutions, found so far.

### 3.7 Serialization/De-serialization of Population and GP-Parameters

Postfix-GP provides GUI for storing and retrieving of the population of individuals as well as GP-parameters from file system. The information about the final population and GP parameters are stored in binary format. The `Serialize()` and `Deserialize()` methods of `BinaryFormatter` class are used for storing and retrieving object information.

### 3.8 Statistics

Postfix-GP provides a facility to collect statistical details at every generation of a GP run. It stores data for individuals of both population and an archive into an excel file. It records following details for individuals of an archive per generation: (i) adjusted fitness value of the best individual of current generation, (ii) size of the best individual (solution), (iii) average adjusted fitness of archive, (iv) average node (element) count of archive, (v) best individual found so far, (vi) size of the best individual found so far, and (vii) Mean Absolute Error (MAE), Normalized Mean Square Error (NMSE), and Correlation Coefficient (CC) values for the best solution found so far. `WritePopulationAnalysisData()` and `WriteArchiveAnalysisData()` methods of `PostfixGPGUI` are used to store these data into the file. The mentioned statistical information are used to produce the following graphs:

- Average adjusted fitness vs generations
- Average node count vs generations
- Best adjusted fitness vs generations
- Average adjusted fitness, average node count, and best adjusted fitness vs generations

### 3.9 Implementation of GUI

The class responsible for providing functionalities of GUI is named `PostfixGPGUI`. It is used to collect training dataset, function and terminal set, and Postfix-GP configuration parameters, provided by the user, and passing these data to Postfix-GP core. The GUI is useful to extend the functionalities of Postfix-GP core. The user can directly load the training dataset from stored comma separated (.csv) file. The `LoadDataFile()` method of `PostfixGPGUI` class is used to load the training dataset. The `LoadDataFile()` internally uses `getColumnNames()`, `getColumn()`, and `getRow()` methods of `FileOperation` class to determine the header of columns, number of columns, and number of rows in training dataset at run time. Then it calls `getData()` method of `FileOperation` class to load the training dataset into rows of `DataGridView`.

At the end of Postfix-GP run, GUI displays the best evolved solutions. The user can visualize the statistical details of any of the evolved solutions by pressing mouse button on the selected solution. The corresponding reference of an instance of `EquationGenome` is obtained from the current instance of `Population`. The `EquationGenome` reference is used to invoke `GetRawfitness()`, `GetAdjustedfitness()`, `GetCorrelationCoefficient()`, `CalculateNMSEFitness()`, `GetValidLength()` methods of `EquationGenome` class which returns the rawfitness, adjustedfitness,  $r$ , NMSE, and size of solution measures for the selected solution. Then, the instance of `Graphs` is created. The values for mentioned measures are passed as arguments of `DrawEquationMethod()` method of `Graphs` class. The `DrawEquation()` and `DrawGraph()` methods of `Graphs` class are useful to plot: (i) solution details and (ii) analysis of results of Postfix-GP run.

## 4 Case Study

We have taken the benchmark symbolic regression problem, solved using GEP approach in [15]. Equation(1) is used to generate a set of twenty one equidistant points in range  $[-10:1:10]$ . These points are used as fitness cases (training dataset).

$$y = 3 * (x + 1)^3 + 2 * (x + 1)^2 + (x + 1), \quad (1)$$

The population size and the number of generations are set to values 50 and 200. The crossover and mutation rates are set to values 0.9 and 0.1. The `MinLength` and `MaxLength` parameters of Postfix-GP are set to values 15 and 35. The terminal set  $T$  consists of the independent variable,  $x$ , and a list of constants,  $T = \{ x, \text{list of}$

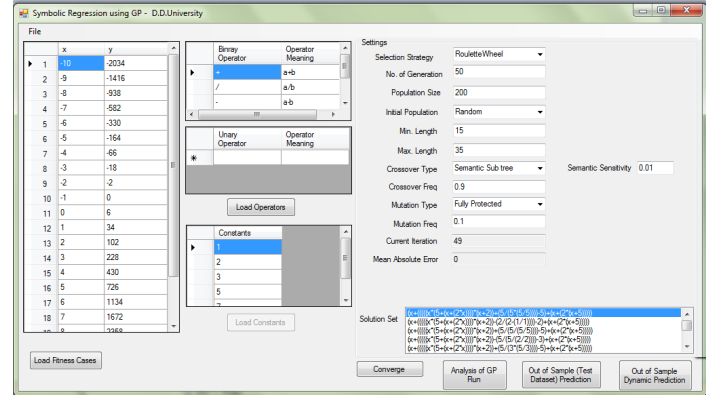


Figure 3: Postfix-GP GUI for Experimentation

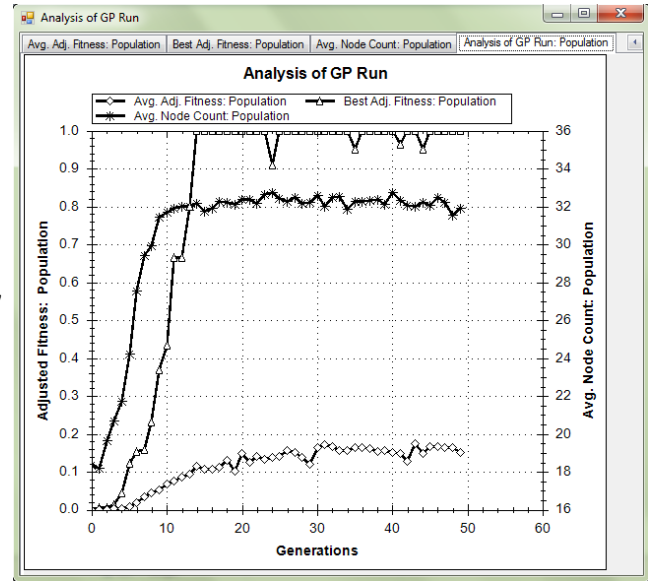


Figure 4: Postfix-GP GUI for analysis of GP run

constants  $\}$ . The selected constants values are  $\{1, 2, 3, 5, 7\}$ . The function set  $F$  consists of  $\{+, -, *, /\}$ .

The left hand side panel of Postfix-GP GUI (Figure 3) provides the facility to load the training dataset, the function set, and the terminal set from the files. The accepted file format is comma separated values (.csv). The training dataset can be loaded from a file by activating *Load Fitness Cases* button of GUI. The function set and the terminal set can be loaded by activating *Load Operators* and *Load Constants* buttons of GUI. The right hand side panel of Postfix-GP GUI (Figure 3) provides facility to configure Postfix-GP parameters.

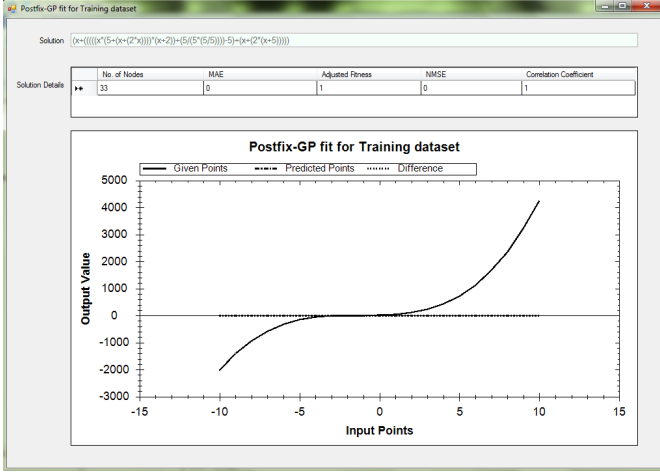


Figure 5: Postfix-GP GUI showing statistical measures for the evolved solution for training data

$$y = (x + (((((x * (5 + (x + (2 * x)))) * (x + 2)) + (5/5 * (5/5)))) - 5) + (x + (2 * (x + 5)))))) \quad (2)$$

The separate GUIs are provided for visualization of: (i) evolved solution with its statistical measures, (ii) Postfix-GP run analysis, and (iii) Prediction of out-of-sample datapoints. Figure 5 presents visual representation of evolved solution with its statistical information. Equation (2) represents one of the best solutions found by Postfix-GP. Simplifying this equation results in equation (1).

Figure 5 depicts the comparison of actual function and the function modeled by the evolved Postfix-GP solution. The evolved solution has structural complexity (number of nodes) of 33. The statistical measures for the evolved solution for training dataset are as follows:  $MAE = 0$ ,  $NMSE = 0$ ,  $AdjustedFitness = 1$ , and  $r = 1$ . Figure 4 presents the visual representation of Postfix-GP run analysis over 50 generations. It displays the plots of the average best fitness vs generation and the average solution size vs generation.

Figure 6 shows out-of-sample (test data) one-step ahead predictions obtained using the evolved solution for the values of  $x$  in the range [11:1:20]. The statistical measures for one-step predictions are as follows:  $MAE = 0$ ,  $NMSE = 0$ ,  $AdjustedFitness = 1$ , and  $r = 1$ . The error of zero value suggest that the one-step ahead predictions are very good.

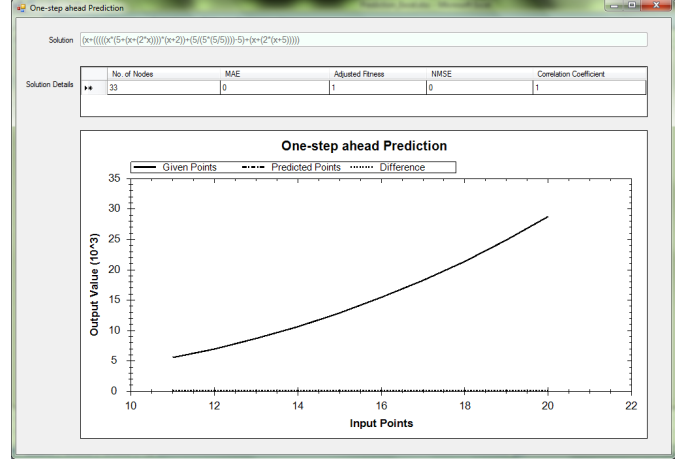


Figure 6: Postfix-GP GUI for one-step prediction of out-of-sample data

## 5 Features Comparison of Postfix-GP with lil-gp, ECJ, and JCLEC

We have compared the features of Postfix-GP with lil-gp [1], ECJ [4], and JCLEC [10] GP systems. We have selected these GP systems because they are open source and freely available. Most of these systems do not provide easy to use functionality to end users. These systems can be used by persons having experience in the field of the evolutionary computing. Many systems are dedicated to a particular flavor of EC and others are generic (can be used for different flavors of EC). For example, lil-gp [1] provides support for GP only, whereas ECJ [4] and JCLEC [10] provide support for Genetic Algorithm (GA) [16], Genetic Programming (GP) [11], and Gene Expression Programming (GEP) [17]. However, for comparison of these systems with Postfix-GP, we have only considered GP flavor of these systems. Table 1 presents the features comparison of Postfix-GP with these systems.

**lil-gp:** lil-gp [1] is a GP toolkit implemented in C programming language. The toolkit is efficient and fast, as it is implemented in C. However, it is difficult to extend the toolkit compare to other object-oriented implementations of GP systems. lil-gp uses only tree structure for an individual representation and provides limited fitness measures. Moreover, the toolkit does not provide graphical user interface to read an input (training) data file. The toolkit uses a parameter file to load GP parameters. The toolkit produces six reporting files (.sys,.gen,.prg,.bst,.his and .stt) that provide the statistical information of the GP run. There are many patches developed by different



Table 1: Comparison of the features of Postfix-GP system with other GP systems

No.	Features	lil-gp	JCLEC	ECJ	Postfix-GP
1	Genome representation	Tree	Tree	Tree	Linear string
2	GP parameters management	Text file (parameter file)	Configuration file	GUI configuration file	GUI
3	Initialization technique	Full, Grow, Half-and-Half	Random, Koza	Random, Koza, PTC1, PTC2	Random, Semantically diverse
4	Crossover operators	Sub-tree	Sub-tree, Tree	Sub-tree	Sub-tree, GA-like one-Point, Semantic sub-tree
5	Limiting size of solution	Number of nodes/Depth limit	Number of nodes/Depth limit	Number of nodes/Depth limit	MinLength, MaxLength
6	Selection schemes	Fitness proportional, Tournament, Random, Best, Worst	Elitist, Fitness proportional, Tournament, Random, MuLambda, Boltzman	Elitist, Fitness proportional, Tournament, Random, MuLambda, Boltzman	Archive, Fitness proportional, Tournament
7	Adding elements to function set	C function (function.c,.h)	Java file, Parameter file	Java file, Parameter file	C# Function, Parameter file
8	Cascading functionality	No	No	No	Yes
9	Sub-trees as solutions	No	No	Yes (GEP)	Yes
10	GP run statistics	Report files	GUI, Text file	.stat file	GUI, Text file
11	GUI for prediction using evolved model	No	No	No	Yes
12	Generic	No	Yes	Yes	No
13	Learning curve	Small	Medium	High	Small
14	GUI for visualization of statistical results	No	Yes	No	Yes
15	Architecture	Procedure oriented	Object oriented	Object oriented	Object oriented
16	Programming language	C	Java	Java	C#

researchers to fix the bugs and to improve the functionality of the basic lil-gp.

**ECJ:** ECJ [4] is a Java based framework for evolutionary computation and genetic programming. ECJ is designed using the object-oriented concepts. Classes of ECJ framework are divided into four layers [4]: (i) utility layer, (ii) basic and custom evolutionary computation layer, (iii) basic and custom genetic programming layer, and (iv) problem layer. As the framework is implemented in Java, it is slower in speed. Moreover, ECJ uses a tree (and not an arrays) to represent an individual, which requires dynamic memory allocation. Thus, the framework consumes large memory. ECJ determines GP parameters from a parameter file. ECJ determines classes to be loaded, the type of problem to be solved, the type of technique to use to solve the problem, and the way to report the statistical results of the run from the parameter file at the run time [4]. This provides easy to extend functionality to the ECJ. ECJ stores statistical information of GP run in a text file. Moreover, it provides flexibility to produce the extra output files through class customization but does not provide a GUI to visualize this information.

**JCLEC:** JCLEC [10] is a Java based framework for evolutionary computation and genetic programming. JCLEC [10] is designed using the object-oriented concepts. The classes of JCLEC framework are divided in three layers: (i) system core, (ii) experiments runner (reads an EA script file, execute all indicated algorithms and produce report files), and (iii) GenLab (a GUI on the top of experiments runner and system core layers, provides functionality to edit the experiment files and to view the GP run results) [10]. GP parameters can be set by the user

either through GenLab GUI or through an XML (configuration) file. However, the structure of configuration files is not user friendly. The framework provides the GUI to visualize statistical information of GP run. The JCLEC framework is easy to extend.

## 6 Conclusions

This paper presented the design and implementation of Postfix-GP framework. The implementation details of Postfix-GP, including an individual representation, different crossover operators, mutations, and selection mechanism were also presented. Postfix-GP provides user interactive GUI for performing different activities. The user can load training dataset, function set, and constants. The user can set the GP parameters through GUI. The evolved solutions with their statistical measures can be visualized through GUI. Moreover, the user can also perform one-step and multi-step predictions using GUI. The evolved solutions can be stored in binary format and can be retrieved later on. The user can also analyze Postfix-GP run through GUI. Postfix-GP as a solution modeling tool was presented by solving symbolic regression problem. Postfix-GP addresses the requirements of ease of use and small learning curve before utilizing it to solve the problems. It was developed to minimize the user's time required to set up and run GP experiments.

## References

- [1] D. Zongker and B. Punch, “lilgp 1.01 genetic programming system,” 1998. [Online]. Available: <http://garage.cse.msu.edu/software/lil-gp/>
- [2] C. Ferreira, *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*, 2nd ed. Springer, May 2006.
- [3] S. Silva and J. Almeida, “Gplab-a genetic programming toolbox for matlab,” pp. 273–278, 2003.
- [4] S. Luke, L. Panait, G. Balan, and Et, “ECJ 16: A Java-based Evolutionary Computation Research System,” <http://cs.gmu.edu/~eclab/projects/ecj/>, 2007.
- [5] C. Gagné and M. Parizeau, “Open BEAGLE: A new C++ evolutionary computation framework,” in *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*. New York: Morgan Kaufmann Publishers, 9-13 July 2002, p. 888. [Online]. Available: <http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/gecco2002/gecco-2002-15.pdf>
- [6] V. Dabhi and S. Vij, “Empirical modeling using symbolic regression via postfix genetic programming,” in *Image Information Processing (ICIIP), 2011 International Conference on*, 2011, pp. 1–6.
- [7] V. Dabhi and S. Chaudhary, “Semantic sub-tree crossover operator for postfix genetic programming,” in *Proceedings of Seventh International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2012)*, ser. Advances in Intelligent Systems and Computing, J. C. Bansal, P. K. Singh, K. Deep, M. Pant, and A. K. Nagar, Eds., vol. 201. Springer India, 2013, pp. 391–402.
- [8] —, “Time series modeling and prediction using postfix genetic programming,” in *Advanced Computing Communication Technologies (ACCT), 2014 Fourth International Conference on*, Feb 2014, pp. 307–314.
- [9] V. K. Dabhi and S. Chaudhary, “Hybrid wavelet-postfix-gp model for rainfall prediction of anand region of india,” *Advances in Artificial Intelligence*, vol. 2014, 2014.
- [10] S. Ventura, C. Romero, A. Zafra, J. A. Delgado, and C. Hervs, “Jelec: a java framework for evolutionary computation,” *Soft Computing*, vol. 12, no. 4, pp. 381–392, 2008.
- [11] J. R. Koza, *Genetic Programming: On the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.
- [12] Microsoft, “Microsoft .net framework software development kit,” 2007. [Online]. Available: <http://msdn.microsoft.com>
- [13] “Zedgraph,” 2008. [Online]. Available: <http://www.sourceforge.net/projects/zedgraph>
- [14] V. K. Dabhi and S. Chaudhary, “Performance comparison of crossover operators for postfix genetic programming,” *International Journal of Metaheuristics*, vol. 3, no. 3, pp. 244–264, 2014.
- [15] X. Li, C. Zhou, W. Xiao, and P. C. Nelson, “Prefix gene expression programming,” in *Late breaking paper at Genetic and Evolutionary Computation Conference (GECCO’2005)*, Washington, D.C., USA, 25-29 Jun. 2005, pp. 25–31.
- [16] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, USA: MIT Press, 1992.
- [17] C. Ferreira, “Gene expression programming: a new adaptive algorithm for solving problems,” *Complex Systems*, vol. 13, no. 2, pp. 87–129, 2001.